CSE544
Data Management

Lectures 3: SQL

Announcements

- Review 1 was due today
- Monday, 1/15: holiday, no class
- Wednesday, 1/17: canceled
- Friday, 1/19: makeup lecture, CSE2-371
- Also Friday, 1/19: review 2 is due

Recap

SQL so far:

SELECT-FROM-WHERE

- FROM: which tables – joins
- WHERE: condition – selections
- SELECT: which attributes – projections

- NULLs…

“A Case Against SQL”

“A Case Against SQL”

Lots of inconsistentices

- NULLs
- Duplicated attributes: SELECT A,A
- Types: 1 = ‘1’
- Corner cases:
  - – Empty string, division by 0, transitivity of =

GROUP-BY

Overview

- Aggregates in SQL:
  - Sum, min, max, count, avg

- select agg(…) → one ouput tuple

- select A,agg(B) … group by A
  → many output tuples

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

Examples

```
SELECT  max(psize)
FROM    Part
```

max
50

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

Examples

```
SELECT  max(psize)
FROM    Part
```

max
50

```
SELECT pcolor, max(psize)
FROM    Part
GROUP BY pcolor
```

color	max
green	12
blue	50
gray	9
red	25
…	

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

Examples

```
SELECT  max(psize)
FROM    Part
```

max
50

```
SELECT pcolor, max(psize)
FROM    Part
GROUP BY pcolor
```

color	max
green	12
blue	50
gray	9
red	25
…	

```
SELECT pcolor, max(psize), sum(psize)
FROM    Part
GROUP BY pcolor
```

. . .

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

Subtleties

```
SELECT pcolor
FROM     Part
GROUP BY pcolor
```

?

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

Subtleties

```
SELECT pcolor
FROM     Part
GROUP BY pcolor
```

Same as distinct

```
SELECT DISTINCT pcolor
FROM     Part
```

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

Subtleties

```
SELECT pcolor
FROM      Part
GROUP BY pcolor
```

⇒

Same as distinct

```
SELECT DISTINCT pcolor
FROM      Part
```

```
SELECT pcolor, pname, max(psize)
FROM      Part
GROUP BY pcolor
```

⇒

?

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

Subtleties

```
SELECT pcolor
FROM      Part
GROUP BY pcolor
```

⇒

Same as distinct

```
SELECT DISTINCT pcolor
FROM      Part
```

```
SELECT pcolor, pname, max(psize)
FROM      Part
GROUP BY pcolor
```

⇒

ERROR

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

Examples

Compute the number of parts supplied by each supplier

```
SELECT sno, count(*)
FROM     Supply
GROUP BY sno
```

Include the names of the suppliers

```
SELECT x.sno, x.sname, count(*)
FROM    Supplier x, Supply y
WHERE  x.sno=y.sno
GROUP BY x.sno, x.sname
```

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

WHERE v.s. HAVING

Compute the total quantity supplied by each supplier in ‘WA’

```
SELECT x.sno, x.sname, sum(y.qty)
FROM    Supplier x, Supply y
WHERE  x.sno=y.sno and x.sstate=‘WA’
GROUP BY x.sno, x.sname
```

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

WHERE v.s. HAVING

Compute the total quantity supplied by each supplier in ‘WA’

```
SELECT x.sno, x.sname, sum(y.qty)
FROM    Supplier x, Supply y
WHERE  x.sno=y.sno and x.sstate=‘WA’
GROUP BY x.sno, x.sname
```

Compute the total quantity supplied by each supplier who supplied > 100 parts

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

WHERE v.s. HAVING

Compute the total quantity supplied by each supplier in ‘WA’

```
SELECT x.sno, x.sname, sum(y.qty)
FROM    Supplier x, Supply y
WHERE  x.sno=y.sno and x.sstate=‘WA’
GROUP BY x.sno, x.sname
```

Compute the total quantity supplied by each supplier who supplied > 100 parts

```
SELECT x.sno, x.sname, sum(y.qty)
FROM    Supplier x, Supply y
WHERE  x.sno=y.sno
GROUP BY x.sno, x.sname
HAVING count(*) > 100
```

Semantics

SELECT $a_1, \ldots, a_k, agg_1, agg_2$
FROM R_1 AS x_1, R_2 AS x_2, …, R_n AS x_n
WHERE condition1($a_1, \ldots, a_k, b_1, \ldots, b_n$)
GROUP BY $a_1, \ldots, a_k$
HAVING condition2($a_1, \ldots, a_k, agg_3, agg_4$)

Semantics

SELECT a_1, …, a_k, agg_1, agg_2
FROM R_1 AS x_1, R_2 AS x_2, …, R_n AS x_n
WHERE condition1(a_1, …, a_k, b_1,…,b_n)
GROUP BY a_1, …, a_k
HAVING condition2(a_1, …, a_k,agg_3,agg_4)

Step 1: FROM-WHERE

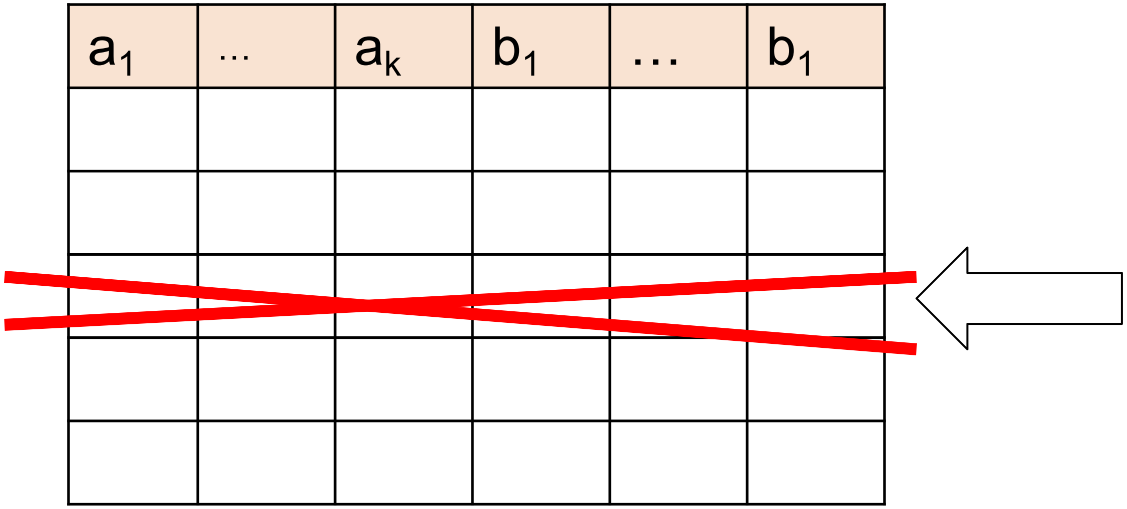
a_1	…	a_k	b_1	…	b_1

Check
WHERE condition1
in each row

Semantics

SELECT $a_1, \ldots, a_k, agg_1, agg_2$
FROM R_1 AS x_1, R_2 AS x_2, …, R_n AS x_n
WHERE condition1($a_1, \ldots, a_k, b_1, \ldots, b_n$)
GROUP BY $a_1, \ldots, a_k$
HAVING condition2($a_1, \ldots, a_k, agg_3, agg_4$)

Step 1: FROM-WHERE



a_1	…	a_k	b_1	…	b_1

Check
WHERE condition1
in each row

Semantics

SELECT $a_1, \ldots, a_k, agg_1, agg_2$
FROM R_1 AS x_1, R_2 AS x_2, …, R_n AS x_n
WHERE condition1($a_1, \ldots, a_k, b_1, \ldots, b_n$)
GROUP BY $a_1, \ldots, a_k$
HAVING condition2($a_1, \ldots, a_k, agg_3, agg_4$)

Step 1: FROM-WHERE

a_1	…	a_k	b_1	…	b_1

Semantics

SELECT $a_1, \ldots, a_k, agg_1, agg_2$
FROM R_1 AS x_1, R_2 AS x_2, …, R_n AS x_n
WHERE condition1($a_1, \ldots, a_k, b_1, \ldots, b_n$)
GROUP BY $a_1, \ldots, a_k$
HAVING condition2($a_1, \ldots, a_k, agg_3, agg_4$)

Step 2: GROUP BY

a_1	…	a_k	b_1	…	b_1
u	…	v			
u		v			
p		q			
p		q			
p		q			

All attributes $a_1, \ldots, a_k$, have the same value inside each group

Semantics

SELECT a_1, …, a_k, agg_1, agg_2
FROM R_1 AS x_1, R_2 AS x_2, …, R_n AS x_n
WHERE condition1(a_1, …, a_k, b_1,…,b_n)
GROUP BY a_1, …, a_k
HAVING condition2(a_1, …, a_k,agg_3,agg_4)

Step 3: HAVING

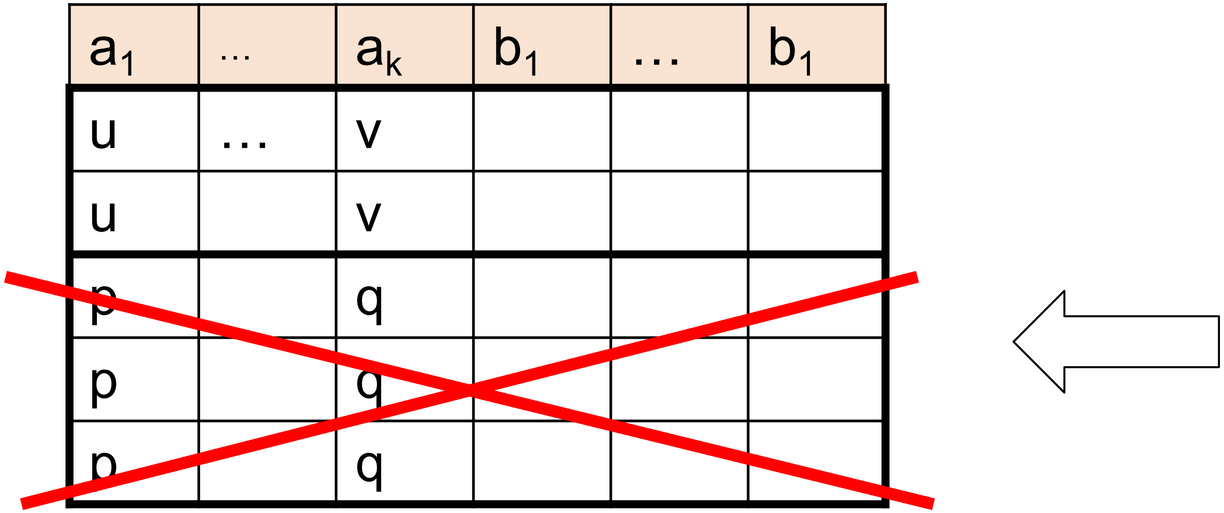
a_1	…	a_k	b_1	…	b_1
u	…	v			
u		v			
p		q			
p		q			
p		q			

Check condition2 in each group

Semantics

SELECT a_1, …, a_k, agg_1, agg_2
FROM R_1 AS x_1, R_2 AS x_2, …, R_n AS x_n
WHERE condition1(a_1, …, a_k, b_1,…,b_n)
GROUP BY a_1, …, a_k
HAVING condition2(a_1, …, a_k,agg_3,agg_4)

Step 3: HAVING



a_1	…	a_k	b_1	…	b_1
u	…	v			
u		v			
~~p~~		~~q~~			
~~p~~		~~q~~			
~~p~~		~~q~~			

Check condition2 in each group

Semantics

SELECT a_1, …, a_k, agg_1, agg_2
FROM R_1 AS x_1, R_2 AS x_2, …, R_n AS x_n
WHERE condition1(a_1, …, a_k, b_1,…,b_n)
GROUP BY a_1, …, a_k
HAVING condition2(a_1, …, a_k,agg_3,agg_4)

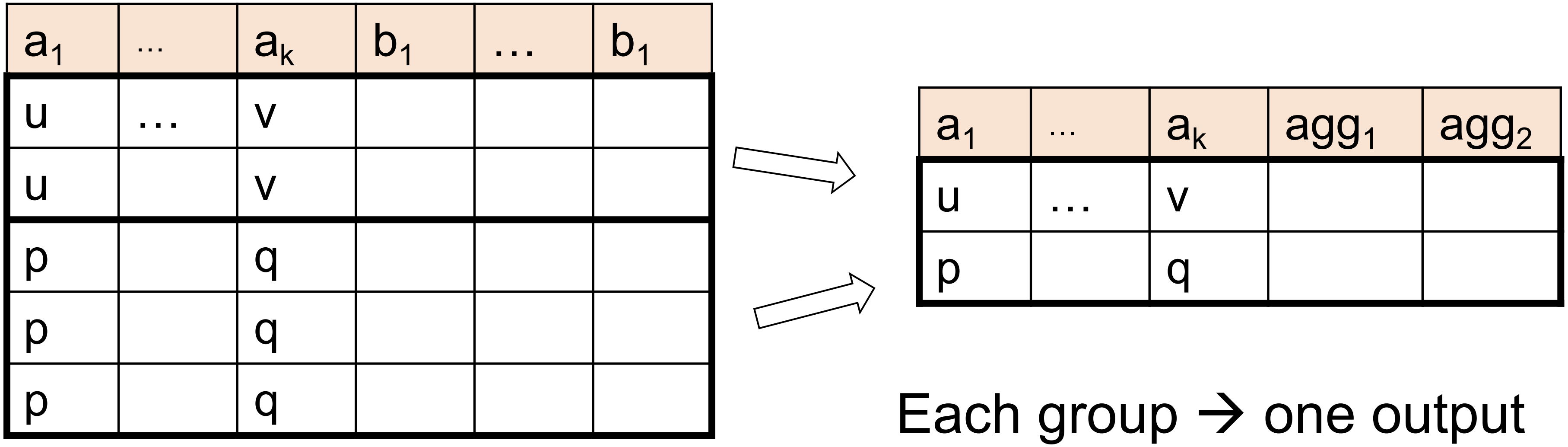
Step 3: HAVING

a_1	…	a_k	b_1	…	b_1
u	…	v			
u		v			
p		q			
p		q			
p		q			

Semantics

SELECT a_1, …, a_k, agg_1, agg_2
FROM R_1 AS x_1, R_2 AS x_2, …, R_n AS x_n
WHERE condition1(a_1, …, a_k, b_1,…,b_n)
GROUP BY a_1, …, a_k
HAVING condition2(a_1, …, a_k,agg_3,agg_4)

Step 4: SELECT



a_1	…	a_k	b_1	…	b_1
u	…	v			
u		v			
p		q			
p		q			
p		q			

a_1	…	a_k	agg_1	agg_2
u	…	v		
p		q		

Each group → one output

Discussion

- GROUP-BY is very versatile in SQL
- No analogous in programming languages: use nested loops instead

```
SELECT x.sno, count(*)
FROM    Supplier x, Supply y
WHERE  x.sno=y.sno
GROUP BY x.sno
```

Discussion

- GROUP-BY is very versatile in SQL
- No analogous in programming languages: use nested loops instead

```
SELECT x.sno, count(*)
FROM    Supplier x, Supply y
WHERE  x.sno=y.sno
GROUP BY x.sno
```

```
for x in Supplier:
  c = 0
  for y in Supply:
     if x.sno==y.sno:
         c = c+1
```

Discussion

- GROUP-BY is very versatile in SQL
- No analogous in programming languages: use nested loops instead

```
SELECT x.sno, count(*)
FROM   Supplier x, Supply y
WHERE  x.sno=y.sno
GROUP BY x.sno
```

```
for x in Supplier:
  c = 0
  for y in Supply:
     if x.sno==y.sno:
         c = c+1
```

- The empty group problem: in SQL no group can be empty. Outer joins!

Empty Groups Problem

- Every group is non-empty
- Consequences:
  - count(*) > 0
  - sum(…) > 0 (assuming numbers are >0)
- Sometimes we want to return 0 counts:
  - Parts that never sold
  - Suppliers that never supplied
- Use outer joins: count(…) skips NULLs

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

Empty Groups Problem

Compute the number of parts supplied by each supplier

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

Empty Groups Problem

Compute the number of parts supplied by each supplier

```
SELECT x.sno, count(*)
FROM    Supplier x, Supply y
WHERE  x.sno=y.sno
GROUP BY x.sno
```

Suppliers who never supplied any part will be missing: count(*) > 0

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

Empty Groups Problem

Compute the number of parts supplied by each supplier

```
SELECT x.sno, count(*)
FROM    Supplier x, Supply y
WHERE  x.sno=y.sno
GROUP BY x.sno
```

Suppliers who never supplied any part will be missing: count(*) > 0

```
SELECT x.sno, count(y.sno)
FROM    Supplier x
  LEFT OUTER JOIN Supply y
ON  x.sno=y.sno
GROUP BY x.sno
```

Now we can get count(*)=0

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

Empty Groups Problem

Compute the number of parts supplied by each supplier

```
SELECT x.sno, count(*)
FROM    Supplier x, Supply y
WHERE  x.sno=y.sno
GROUP BY x.sno
```

Suppliers who never supplied any part will be missing: count(*) > 0

```
SELECT x.sno, count(y.sno)
FROM    Supplier x
  LEFT OUTER JOIN Supply y
ON  x.sno=y.sno
GROUP BY x.sno
```

Now we can get count(*)=0

Cannot write count(*). Why?

OUTER JOIN

Outer Joins

- A join returns only those outputs that have a tuple from each of the input tables
- Sometimes we want to include tuples from one table without a match from the other table:

Outer Join

```
Product(name, category)
Purchase(prodName, store)
```

prodName is foreign Key

Outer joins

Retrieve all product names, categories, and stores where they were purchased. Include products that never sold

```
Product(name, category)
Purchase(prodName, store)
```

prodName is foreign Key

Outer joins

Retrieve all product names, categories, and stores where they were purchased. Include products that never sold

```
SELECT x.name, x.category, y.store
FROM   Product x, Purchase y
WHERE  x.name = y.prodName
```

```
Product(name, category)
Purchase(prodName, store)
```

prodName
is foreign Key

Outer joins

Retrieve all product names, categories, and stores where they were purchased.
Include products that never sold

```
SELECT x.name, x.category, y.store
FROM   Product x, Purchase y
WHERE  x.name = y.prodName
```

Product

Name	Category
Gizmo	gadget
Camera	Photo
OneClick	Photo

Purchase

ProdName	Store
Gizmo	Wiz
Camera	Ritz
Camera	Wiz

```
Product(name, category)
Purchase(prodName, store)
```

prodName is foreign Key

Outer joins

Retrieve all product names, categories, and stores where they were purchased.
Include products that never sold

```
SELECT x.name, x.category, y.store
FROM   Product x, Purchase y
WHERE  x.name = y.prodName
```

Product

Name	Category
Gizmo	gadget
Camera	Photo
OneClick	Photo

missing

Purchase

ProdName	Store
Gizmo	Wiz
Camera	Ritz
Camera	Wiz

Output

Name	Category	Store
Gizmo	gadget	Wiz
Camera	Photo	Ritz
Camera	Photo	Wiz

```
Product(name, category)
Purchase(prodName, store)
```

prodName is foreign Key

Outer joins

Retrieve all product names, categories, and stores where they were purchased. Include products that never sold

```
SELECT x.name, x.category, y.store
FROM   Product x LEFT OUTER JOIN Purchase y
ON     x.name = y.prodName
```

Product

Name	Category
Gizmo	gadget
Camera	Photo
OneClick	Photo

Purchase

ProdName	Store
Gizmo	Wiz
Camera	Ritz
Camera	Wiz

Output

Name	Category	Store
Gizmo	gadget	Wiz
Camera	Photo	Ritz
Camera	Photo	Wiz
OneClick	Photo	NULL

Now it's present

Left Outer Join (Details)

from R left outer join S on C1 where C2

1. Compute cross product R×S
2. Filter on C1
3. Add all R records without a match
4. Filter on C2

Left Outer Join (Details)

```
select ...
from    R left outer join S on C1
where   C2
```

```
Tmp = {}
for x in R do                          // left outer join using C1
    for y in S do
        if C1 then Tmp = Tmp ∪ {(x,y)}
for x in R do
    if not (x in Tmp) then Tmp = Tmp ∪ {(x,NULL)}

Answer = {}                            // apply condition C2
for (x,y) in Tmp if C2 then Answer = Answer ∪ {(x,y)}
return Answer
```

```
Product(name, category)
Purchase(prodName, store, price)
```

prodName
is foreign Key

ON v.s. WHERE

- Outer join condition in the ON clause
- Different from the WHERE clause
- Compare:

```
SELECT x.name, y.store
FROM   Product x
LEFT OUTER JOIN Purchase y
ON     x.name = y.prodName
   AND y.price < 10
```

```
SELECT x.name, y.store
FROM   Product x
LEFT OUTER JOIN Purchase y
ON     x.name = y.prodName
WHERE y.price < 10
```

```
Product(name, category)
Purchase(prodName, store, price)
```

prodName is foreign Key

ON v.s. WHERE

- Outer join condition in the ON clause
- Different from the WHERE clause
- Compare:

```
SELECT x.name, y.store
FROM   Product x
LEFT OUTER JOIN Purchase y
ON     x.name = y.prodName
   AND y.price < 10
```

Includes products that were never purchased with price < 10

```
SELECT x.name, y.store
FROM   Product x
LEFT OUTER JOIN Purchase y
ON     x.name = y.prodName
WHERE y.price < 10
```

```
Product(name, category)
Purchase(prodName, store, price)
```

prodName is foreign Key

ON v.s. WHERE

- Outer join condition in the ON clause
- Different from the WHERE clause
- Compare:

```
SELECT x.name, y.store
FROM   Product x
LEFT OUTER JOIN Purchase y
ON     x.name = y.prodName
   AND y.price < 10
```

Includes products that were never purchased with price < 10

```
SELECT x.name, y.store
FROM   Product x
LEFT OUTER JOIN Purchase y
ON     x.name = y.prodName
WHERE y.price < 10
```

Includes products that were never purchased, *then* checks price <10

```
Product(name, category)
Purchase(prodName, store, price)
```

prodName is foreign Key

ON v.s. WHERE

- Outer join condition in the ON clause
- Different from the WHERE clause
- Compare:

```
SELECT x.name, y.store
FROM   Product x
LEFT OUTER JOIN Purchase y
ON     x.name = y.prodName
   AND y.price < 10
```

Includes products that were never purchased with price < 10

```
SELECT x.name, y.store
FROM   Product x
LEFT OUTER JOIN Purchase y
ON     x.name = y.prodName
WHERE y.price < 10
```

Includes products that were never purchased, *then* checks price <10

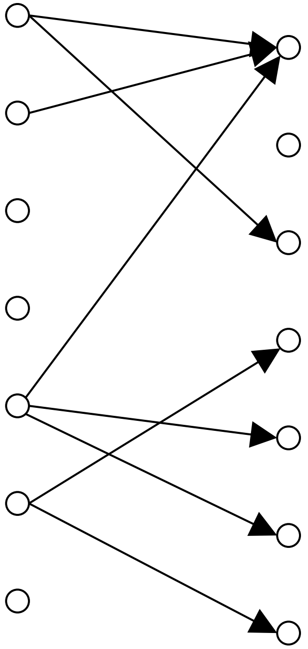
Same as inner join!

Joins

- **Inner join** = includes only matching tuples (i.e. regular join)
- **Left outer join** = includes everything from the left
- **Right outer join** = includes everything from the right
- **Full outer join** = includes everything

Discussion

- LEFT OUTER JOIN is useful for one-to-many relationships
- Interaction between different types of joins makes optimization difficult



Subqueries

Subqueries

- A subquery is a self-contained SQL query that occurs inside another query
- The subquery can be any of these clauses:
  - SELECT
  - FROM
  - WHERE
  - HAVING

```
Product (pname,  price, cid)
Company(cid, cname, city)
```

Subqueries in SELECT

For each city, find the number of products manufactured in that city

```
Product (pname,  price, cid)
Company(cid, cname, city)
```

Subqueries in SELECT

For each city, find the number of products manufactured in that city

```
SELECT DISTINCT x.city, (SELECT count(*)
                         FROM Product y
                         WHERE x.cid = y.cid)
FROM Company x
```

```
Product (pname,  price, cid)
Company(cid, cname, city)
```

Subqueries in SELECT

For each city, find the number of products manufactured in that city

```
SELECT DISTINCT x.city, (SELECT count(*)
                         FROM Product y
                         WHERE x.cid = y.cid)
FROM Company x
```

This is not nice SQL style. Unnest the query to:

```
SELECT x.city, count(*)
FROM Company x, Product y
WHERE x.cid=y.cid
GROUP BY x.city
```

```
Product (pname,  price, cid)
Company(cid, cname, city)
```

Subqueries in SELECT

For each city, find the number of products manufactured in that city

```
SELECT DISTINCT x.city, (SELECT count(*)
                         FROM Product y
                         WHERE x.cid = y.cid)
FROM Company x
```

This is not nice SQL style. Unnest the query to:

```
SELECT x.city, count(*)
FROM Company x, Product y
WHERE x.cid=y.cid
GROUP BY x.city
```

Correction:

```
SELECT x.city, count(y.cid)
FROM Company x LEFT OUTER JOIN
Product y ON x.cid=y.cid
GROUP BY x.city
```

```
Product (pname,  price, cid)
Company(cid, cname, city)
```

Subqueries in FROM

List all products manufactured in Seattle
and their manufacturers names

```
SELECT x.cname, y.pname
FROM (SELECT * FROM Company WHERE city=‘Seattle’) x, Product y
WHERE x.cid=y.cid
```

```
Product (pname,  price, cid)
Company(cid, cname, city)
```

Subqueries in FROM

List all products manufactured in Seattle
and their manufacturers names

```
SELECT x.cname, y.pname
FROM (SELECT * FROM Company WHERE city=‘Seattle’) x, Product y
WHERE x.cid=y.cid
```

This is not nice SQL style. Unnest the query to:

```
SELECT x.cname, y.pname
FROM x, Product y
WHERE x.cid=y.cid and x.city=‘Seattle’
```

```
Product (pname,  price, cid)
Company(cid, cname, city)
```

Subqueries in WHERE

Find all companies that make some products with price < 200

Existential quantifiers

```
Product (pname,  price, cid)
Company(cid, cname, city)
```

Subqueries in WHERE

Find all companies that make some products with price < 200

Existential quantifiers

Using EXISTS:

```
SELECT C.cid, C.cname
FROM    Company C
WHERE  EXISTS (SELECT *
                FROM Product P
               WHERE C.cid = P.cid and P.price < 200)
```

```
Product (pname,  price, cid)
Company(cid, cname, city)
```

Subqueries in WHERE

Find all companies that make some products with price < 200

Existential quantifiers

Using IN

```
SELECT C.cid, C.cname
FROM    Company C
WHERE C.cid IN (SELECT P.cid
                FROM Product P
                WHERE P.price < 200)
```

```
Product (pname,  price, cid)
Company(cid, cname, city)
```

Subqueries in WHERE

Find all companies that make some products with price < 200

Existential quantifiers

Using ANY:

```
SELECT C.cid, C.cname
FROM    Company C
WHERE 200 > ANY (SELECT price
                 FROM Product P
                 WHERE P.cid = C.cid)
```

```
Product (pname,  price, cid)
Company(cid, cname, city)
```

Subqueries in WHERE

Find all companies that make some products with price < 200

Existential quantifiers

Now let's unnest it:

```
SELECT DISTINCT C.cid, C.cname
FROM    Company C, Product P
WHERE  C.cid= P.cid and P.price < 200
```

Existential quantifiers are easy ! ☺

```
Product (pname,  price, cid)
Company(cid, cname, city)
```

Subqueries in WHERE

Find all companies that make only products with price < 200

```
Product (pname,  price, cid)
Company(cid, cname, city)
```

Subqueries in WHERE

Universal quantifiers

Find all companies that make only products with price < 200

same as:

Find all companies where all products have price < 200

```
Product (pname,  price, cid)
Company(cid, cname, city)
```

Subqueries in WHERE

Universal quantifiers

Find all companies that make only products with price < 200

same as:

Find all companies where all products have price < 200

Universal quantifiers are hard ! ☹

```
Product (pname,  price, cid)
Company(cid, cname, city)
```

Subqueries in WHERE

1. Find *the other* companies: i.e. s.t. some product ≥ 200

```
SELECT C.cid, C.cname
FROM    Company C
WHERE  C.cid IN (SELECT P.cid
                           FROM Product P
                           WHERE P.price >= 200)
```

```
Product (pname,  price, cid)
Company(cid, cname, city)
```

Subqueries in WHERE

1. Find *the other* companies: i.e. s.t. some product ≥ 200

```
SELECT C.cid, C.cname
FROM    Company C
WHERE  C.cid IN (SELECT P.cid
                 FROM Product P
                 WHERE P.price >= 200)
```

2. Find all companies s.t. all their products have price < 200

```
SELECT C.cid, C.cname
FROM    Company C
WHERE  C.cid NOT IN (SELECT P.cid
                     FROM Product P
                     WHERE P.price >= 200)
```

```
Product (pname,  price, cid)
Company(cid, cname, city)
```

Subqueries in WHERE

Universal quantifiers

Find all companies that make only products with price < 200

same as:

Find all companies where all products have price < 200

Using EXISTS:

```
SELECT C.cid, C.cname
FROM    Company C
WHERE NOT EXISTS (SELECT *
                  FROM Product P
                  WHERE P.cid = C.cid and P.price >= 200)
```

```
Product (pname,  price, cid)
Company(cid, cname, city)
```

Subqueries in WHERE

Universal quantifiers

Find all companies that make only products with price < 200

same as:

Find all companies where all products have price < 200

Using ALL:

```
SELECT C.cid, C.cname
FROM    Company C
WHERE 200 > ALL  (SELECT price
                  FROM Product P
                  WHERE P.cid = C.cid)
```

Discussion

- SQL has a natural semantics based on the existential quantifier
- For a universal quantifier, we have several options:
  - Use double negation: $\forall x P(x) = \neg\neg\forall x P(x) = \neg\exists x \neg P(x)$
  - Use aggregates: count(*)=0. But remember empty groups!

Finding Witnesses a.k.a. ARGMAX

Argmax

- Find the city with the largest population
- Find product/products with largest price
- Common theme: we want the witness for that largest value
- SQL does not have ARGMAX; there several ways around that.

```
Product (pname,  price, cid)
Company(cid, cname, city)
```

ARGMAX

For each city, find the name of the most expensive product manufactured in that city

```
Product (pname,  price, cid)
Company(cid, cname, city)
```

ARGMAX

For each city, find the name of the most expensive product manufactured in that city

Solution 1: compute (city,max(price)) in subquery

```
SELECT DISTINCT x.city, y.pname
FROM     Company x, Product y,
         (SELECT u.city, max(v.price) as p
          FROM Company u, Product v
          WHERE u.cid = v.cid) z
WHERE x.cid = y.cid
      and x.city=z.city
      and y.price=z.p
```

```
Product (pname,  price, cid)
Company(cid, cname, city)
```

ARGMAX

For each city, find the name of the most expensive product manufactured in that city

Solution 2: use NOT EXISTS

```
SELECT DISTINCT x.city, y.pname
FROM    Company x, Product y
WHERE x.cid = y.cid
      and NOT EXISTS (SELECT * FROM Company u, Product v
                           WHERE u.cid=v.cid
                                 and x.city=u.city
                                 and x.city=u.city
                                 and v.price > y.price)
```

```
Product (pname,  price, cid)
Company(cid, cname, city)
```

ARGMAX

For each city, find the name of the most expensive product manufactured in that city

Solution 3 my favorite☺: use GROUP-BY and HAVING

```
SELECT x.city, y.pname
FROM     Company x, Product y, Company u, Product v
WHERE x.cid = y.cid and u.cid = v.cid
        and x.city=u.city
GROUP BY x.city, y.pname
HAVING y.price >= max(v.price)
```

Summary

- Topics we covered should be enough to write almost any query
- Be mindful of what the optimizer can do:
  - select-from-where-groupby can be optimized efficiently
  - Complex, nested queries, less so
- What we left out:
  - Recursion (→ datalog), window operations